# SISTEMA DE COMERCIO ELECTRÓNICO SEGURO

TRABAJO FIN DE GRADO

**Autor: Borja Marabini Vega**
**Matrícula: p070009**
**DNI:53731897X**

## Índice

# 1. Introducción

A lo largo de esta memoria vamos a describir los diferentes elementos necesario para la construcción de un sistema de comercio electrónico seguro. Para ello será necesario garantizar la confidencialidad de la información en la transacción comercial.

A modo de complementar los conceptos de este proyecto se han añadido 5 anexos, que incluyen una guía de usuario para poder utilizar este sistema.

## 2. Descripción General

El trabajo de Fin de Grado realizado es un sistema de comercio electrónico seguro que consta de tres entidades que serán Cliente, Vendedor y Banco.

El fin de este proyecto es conseguir que el intercambio de información que se realiza entre las distintas entidades a la hora de comprar un producto, se haga de una manera segura y confidencial.

Los objetivos de este proyecto serán:

- Con el protocolo SSL garantizar la confidencialidad y autenticación de los usuarios.
- Infraestructura TCP/IP para poder conectar las distintas entidades.
- Desarrollo de un código para construir las firmas asimétricas de datos.
- Implementar las transacciones que se realizaran entre las entidades para poder comprar un producto.
- Comprobar que al comprar un producto se realizan los intercambios necesarios entre las entidades.
- A la hora de realizar la compra del producto, la tarjeta que utilizará el cliente tendrá una identidad desconocida para el comerciante. Solo el Banco podrá identificarla.

El proyecto se desarrollará utilizando las librerías de Java JCE (Java Cryptography Extension) y JSSE (Java Secure Sockets Extension).

# 3.Infraestructura de Seguridad

Para realizar este proyecto, el primer paso ha sido crear la infraestructura de seguridad. Debido a que utilizamos el protocolo SSL (Anexo4) es necesario crear los certificados y los almacenes donde irán alojados esos certificados para las tres entidades de nuestro proyecto.

Los certificados verificarán la identidad de cada entidad, sino al realizar las conexiones no podríamos autenticar quien o que se conecta. Los almacenes contendrán los certificados y las claves necesarias para la correcta autenticación de las distintas entidades.

## 3.1 Almacenes

Para este sistema de comercio, vamos a tener dos tipos de almacenes para cada entidad. Que serán:

1. **"Private Key entries":** Contiene información de claves criptográfica muy sensible, que es almacenada en formato protegido para evitar accesos no autorizados. Típicamente las claves almacenadas en esta entrada son claves privadas acompañadas por el certificado asociado a la clave pública.

2. **"Trusted Certificate Entries":** Contiene un certificado simple de clave pública perteneciente a otra entidad. Es llamado certificado seguro "Trusted certificate" ya que el propietario del almacén es el que asegura que la clave pública del certificado pertenece a la identidad que aparece en el propietario del certificado. El emisor del certificado avala esto con su firma. Dicho de otro modo el propietario del almacén se fía de la entidad que firma el certificado.

Una vez identificados los diferentes almacenes existentes, tendremos uno de cada tipo para cada entidad. Siendo los nombres:

1. Del tipo **Private Key Entries** :
   - Cliente: AlmacenCL
   - Tienda: AlmacenSR
   - Banco: AlmacenBA

2. Del tipo **Trusted Certificate Entries**:
   - Cliente: AlmacenCLTrust
   - Tienda: AlmacenSRTrust
   - Banco: AlmacenBATrust

## 3.2 Certificados

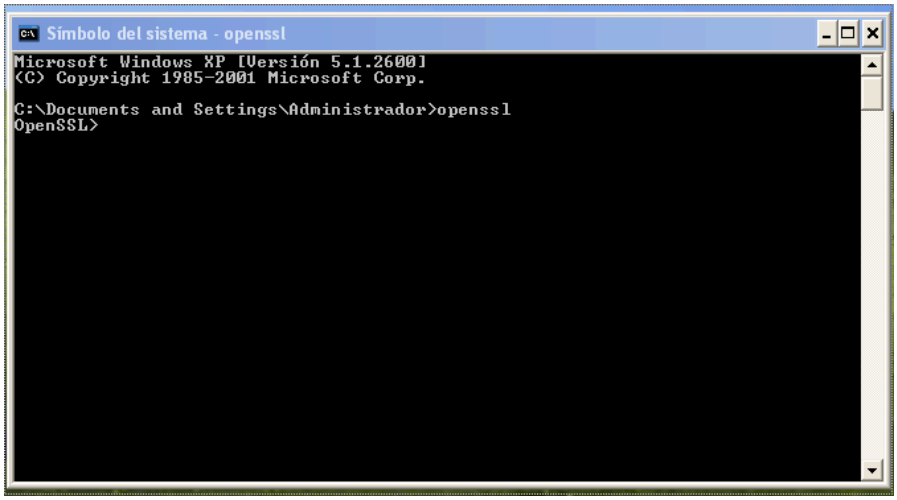
Tras la creación de los almacenes, vamos a ver como crear los certificados de cada entidad, una autoridad de certificación y firmar con esta ultima los certificados.

### 3.2.1 Creación de un Certificado de CA

En primer lugar hemos creado una autoridad de certificación(CA), que es una entidad de confianza responsable de emitir y revocar los certificados utilizados en la firma electrónica, para la cual se utiliza la criptografía de clave pública.

En un caso real ,se debía crear una por entidad, pero para mayor entendimiento y simplicidad del proyecto, se creo una sola CA.El método de creación de esta autoridad de certificación lo hemos realizado en el entorno OpenSSL (Anexo 2) , realizando los siguientes pasos:

1. Nos conectamos al entorno OpenSSL, para ello abriremos la consola de Windows y ejecutaremos el comando : openssl. De esta manera accederemos al entorno OpenSSL.




2. Una vez estemos en el entorno, generamos de un par clave pública-privada.

   - **Comando**: genrsa –out CAClavePrivada.pem 4096

3. Ahora generamos el certificado CA

   - **Comando:** req –new –x509 –days 3650 –key CAClavePrivada.pem –out CACertificado.pem

4. Por ultimo, creamos en formato p12, con la clave privada incorporada protegida con una clave cifrada.

   - **Comando:** pkcs12 –export –in –CACertificado.pem –inkey CAClavePrivada.pem –out CACertificado.p12

### 3.2.2 Firma de los Certificados por una CA

Ya tendríamos generada la autoridad de Certificación, por lo que pasaremos a firmar los certificados de las tres entidades que forman parte del proyecto. Mostraremos el ejemplo de una entidad, usaremos de ejemplo al Cliente , ya que sería el mismo método para todas , simplemente modificando los nombres de los archivos y almacenes.

1. Metemos el certificado de la CA en el "AlmacenCLTrust" , para ello utilizaremos la herramienta keytool.

   - **Comando:** keytool -import -alias CertificadoCA -file cacert.pem –keystoreAlmacenCLTrust

2. Generamos un par de claves pública y secreta junto con el certificado asociado y lo metemos en el almacén "AlmacenCL" con el Alias "CertificadoCL".

   - **Comando:** keytool -genkey -alias CertificadoCL -keyalg RSA -validity "100" –keystore AlmacenCL -keypass oooooo -storepass oooooo

     - **Nota:** Al ejecutar este comando se crea el certificado del cliente, y nos ira pidiendo los datos a rellenar, como nombre y apellidos, nombre de la organización…

3. Ahora vamos a crear un fichero de solicitud de firma "CL.csr". La solicitud de firma CSR (Certification Signing Request) es un proyecto de certificado que requiere la firma de una Autoridad de Certificación.

   - **Comando:** keytool -certreq -alias CertificadoCL -keystore AlmacenCL -file CL.csr

4. En este paso , es necesario volver a utilizar plataforma OpenSSL para generar la firma de la firma de la CA. Nótese que como parámetros de entrada utilizamos el fichero con la petición de firma "CL.csr" y los ficheros con el certificado de la CA "cacert.pem" y la clave privada de la CA "cakey.pem". El resultado es el fichero "CertificadoCL_firmado.crt" que finalmente el certificado original del cliente firmado ahora por la CA.

   - **Comando:** "Openssl> x509 -req -days 365 -in CL.csr -CA cacert.pem -CAkey cakey.pem -set_serial 02 -out CertificadoCL_firmado.crt"

5. Nos salimos de nuevo del entorno OpenSSL para seguir utilizando la herramienta keytool. Este paso es la importación del certificado de la CA en el almacén del cliente "AlmacenCL".

   - **Comando:** keytool -import -alias CertificadoCA -file cacert.pem -keystore AlmacenCL

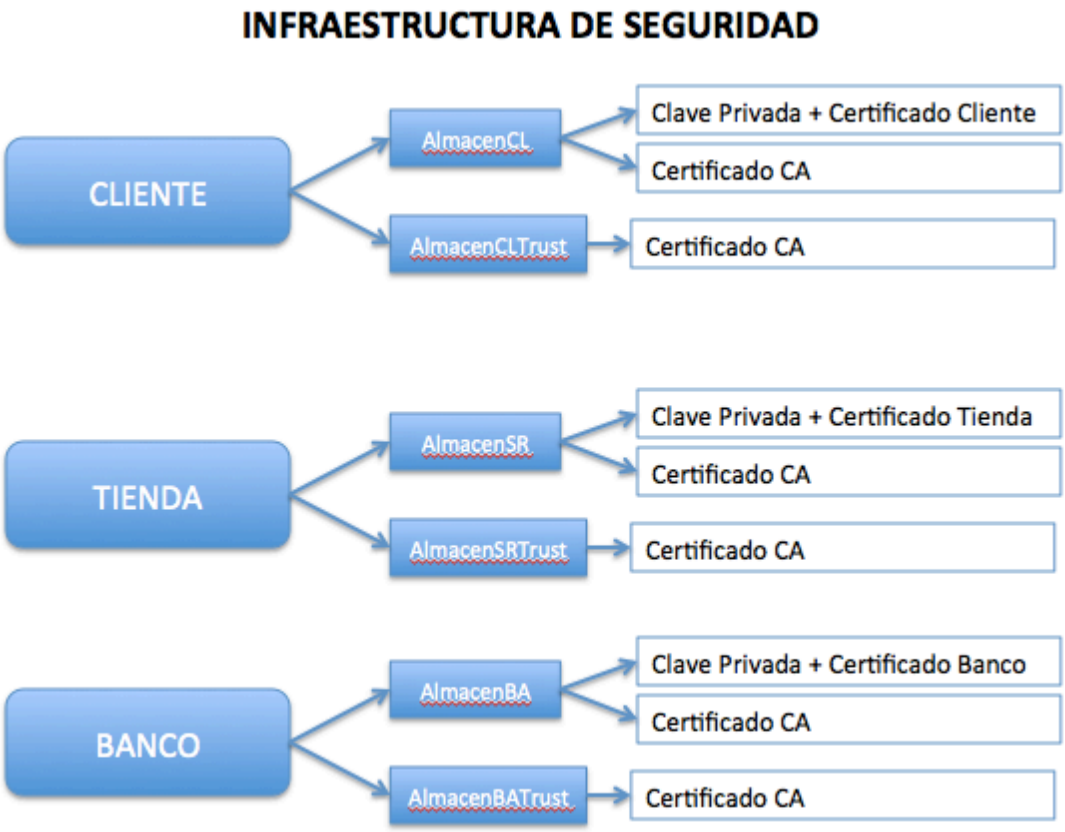
6. El último paso sería la importación del certificado firmado "CertificadoCL_firmado.crt" en el "AlmacenCL". Nótese que la importación se hace con el mismo Alias (CertificadoCL) que teníamos en el Paso 2 cuando el cliente generó su certificado. Esto significa que lo que hacemos es sobrescribir esta entrada con el nuevo certificado ya firmado

    - **Comando:** keytool -import -alias CertificadoCL –file CertificadoCL_firmado.crt –keystore AlmacenCL

### 3.3 Esquema Infraestructura de Seguridad

Para tener una visión mas clara de la infraestructura de seguridad comentada en los anteriores puntos, podemos observar el esquema de cómo quedaría.

Hay que tener en cuenta que todos lo certificados de las entidades están firmado por la CA generada.



# 4. Infraestructura Comunicaciones

Una vez ya creada la infraestructura de seguridad, podemos pasar a la de comunicaciones. Como ya se ha comentado en puntos anteriores, se va a realizar con el protocolo SSL (Anexo 4).

Para que nuestra aplicación desarrollada en código java pueda utilizar el protocolo SSL deben invocar las clases de sockets modificadas incorporando el protocolo SSL. Estas son entre otras las clases "SSLSocket" y "SSLServerSockets" que hemos utilizado en este trabajo de fin de grado.

También de cara a gestionar los diferentes intercambios entre las entidades , vamos a describir como se manejan esos flujos de datos en Java.

Otro aspecto importante es el cifrado de datos a través del cifrador RSA.

## 4.1 Sockets SSL

La clase "SSLSocket" sirve para establecer la conexión y los "SSLServerSockets" junto con los métodos de autenticación y de aceptar la conexión para esperar la llegada de conexiones.

Para implementar los mecanismos de seguridad propios del protocolo SSL se utilizan certificados de clave pública. En java estos certificados normalmente se obtienen de los almacenes de claves que hemos visto en el punto anterior.

Para ello en cada entidad hay que cargar los almacenes, en java se hace a través de las "Java System Properties":

A modo de ejemplo en el Cliente se realizo este paso con el siguiente código:

- System.*setProperty*("javax.net.ssl.keyStore","AlmacenCL");
- System.*setProperty*("javax.net.ssl.keyStorePassword","oooooo");
- System.*setProperty*("javax.net.ssl.trustStore","AlmacenCLTrust");

Una vez tengamos los almacenes cargados, pasamos a crear las conexiones necesarias en cada entidad para establecer las comunicaciones.

## 4.2 Envío y Recepción de Datos

Una vez se establecen a través de los sockets las comunicaciones entre las distintas entidades, es necesario saber como se van a transmitir los datos que hagan que las transacciones del comercio sean completas.

Para ello se utiliza los objetos "OutputStream" y "InputStream" y con los métodos "getOutputStream" y "getInputStream" de la clase "Socket" podremos realizar los envíos y recepciones de datos necesarios entre el Cliente , la Tienda y el Banco.

Es reseñable, que aparte de los objetos "OutputStream" y "InputStream" , hay objetos mas específicos , por ejemplo para datos hemos utilizado "DataOutputStream" y "DataInputStream" .

Un ejemplo de uso de estos objetos, vamos a describir como recibe la Tienda el producto enviado por el Cliente:

1. Obtenemos el flujo que entra a nuestra entidad. Se hace de la siguiente manera:
   - DataInputStream Flujo_eCl = **new** DataInputStream(sslsocket.getInputStream());

2. Ahora del flujo de entrada leemos los enteros que hemos recibido.
   - **Int** prueba;
   - prueba = Flujo_eCl.readInt();

## 4.3 Cifrado de datos

En este proyecto , para garantizar la confidencialidad de los datos, en concreto del número de tarjeta, se ha procedido a cifrar esa información.

Por este motivo hemos incluido en el código un cifrador, en concreto hemos utilizado el RSA, para ello ha sido necesario añadir un provider nuevo. Esto es debido a que en la librerías que hemos utilizado para la criptografía , la JCE (Java Cryptography Extension) no soporta por defecto el RSA.

El nuevo provider es el BouncyCastle, con él añadido ya se ha podido realizar tanto el cifrado del número de tarjeta en el Cliente como el descifrado en el Banco para poder realizar la transacción bancaria.

## 4.4 Comunicaciones

En los siguientes subapartados vamos a detallar los flujos de comunicaciones y datos que habrá en cada uno de las entidades.

### 4.4.1 Cliente

El Cliente solo se comunica con la Tienda, para ello creará un socket indicando la dirección y puerto de dicha entidad. A lo largo de todo el proyecto la dirección ha sido omitida ya que ha sido un proyecto a nivel local, por lo que dicha dirección será siempre "LocalHost". En cambio el puerto si que es necesario establecerlo, y hemos trabajado siempre con el 21002.

Los flujos de comunicación que tendrá el cliente son:

1. Crear el socket con la dirección localhost y puerto 21002.

2. Si se confirma la conexión , enviará el producto a comprar.

3. Recibirá el número de pedido y el certificado del banco, este ultimo necesario para el paso posterior.

4. Enviaremos el número de tarjeta, que cifraremos con la clave pública del certificado del banco recibido en el paso 3. Utilizamos el del banco ya que será quien tenga que visualizar esta información.

5. Por último, recibirá si todo ha ido correctamente, la confirmación del pedido.

### 4.4.2 Tienda

La Tienda tiene comunicación tanto con el Banco como con el Cliente. Al igual que en el Cliente solo es necesario establecer los puertos de los sockets ya que la dirección por defecto será “LocalHost”.

Los flujos de comunicación serán:

1. Creamos un  serversocket en el puerto 21002 , esperando conexiones.

2. Cuando llega alguna conexión, si la autenticación es correcta se acepta la conexión.

3. Recibe el producto que el cliente quiere comprar.

4. Crea un socket con el puerto 50002 , correspondiente al Banco.

5. Genera el número de pedido asociado a esa transacción y se lo envía al cliente junto con el certificado del Banco , que hemos obtenido al establecer conexión con esta entidad.

6. Recibe el número de tarjeta cifrado del Cliente y envía ese numero de tarjeta cifrado al Banco.

7. En caso de que todo sea correcto, envía al Cliente que el pedido ha sido aceptado.

### 4.4.3 Banco

El Banco solo recibe información de la Tienda, y las diferentes comunicaciones que tendrá con la Tienda son:

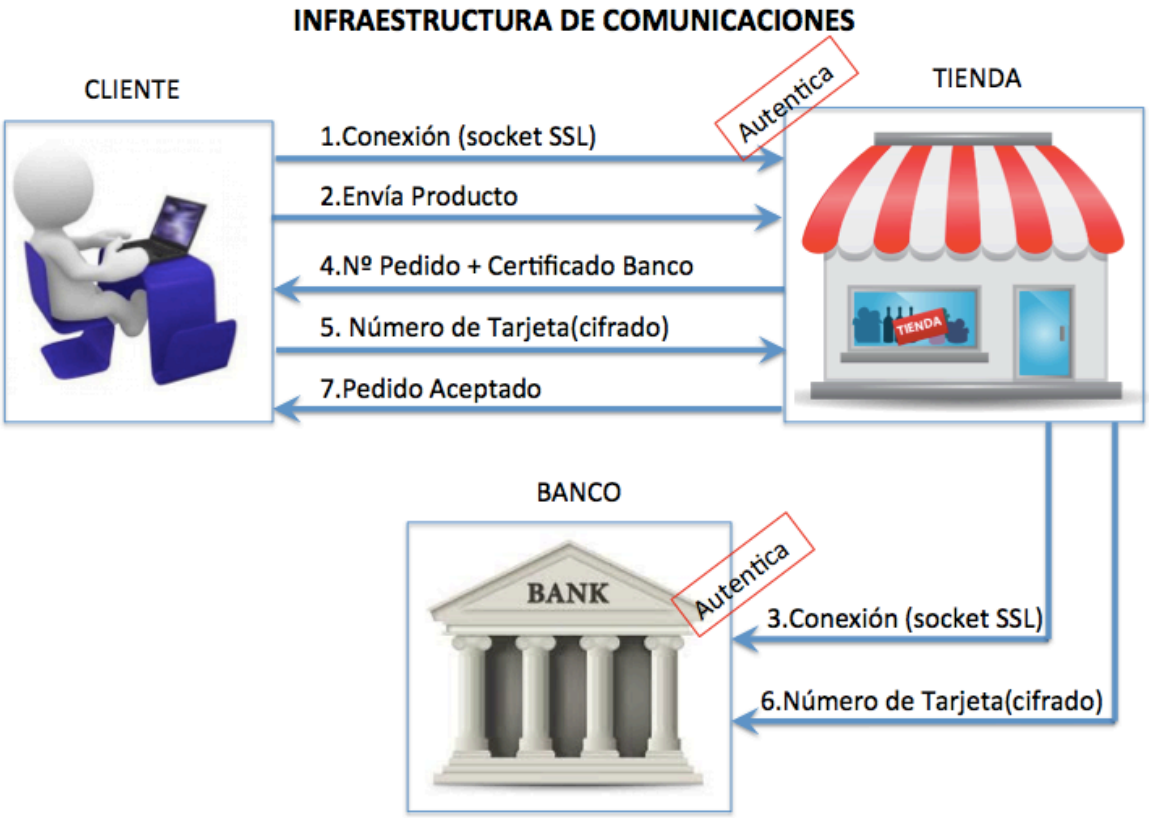
1. Creará un serversocket en el puerto 50002, esperando conexiones

2. Cuando llega alguna conexión, si la autenticación es correcta se acepta la conexión.

3. Recibe el número de tarjeta del Cliente, lo descifra con su clave privada , obteniendo ese dato para poder realizar la transacción económica.

## 4.5 Esquema Arquitectura de Comunicaciones

De cara a reunificar los conceptos , en la siguiente figura podemos observar un esquema de cómo sería nuestro sistema de comercio electrónico seguro.



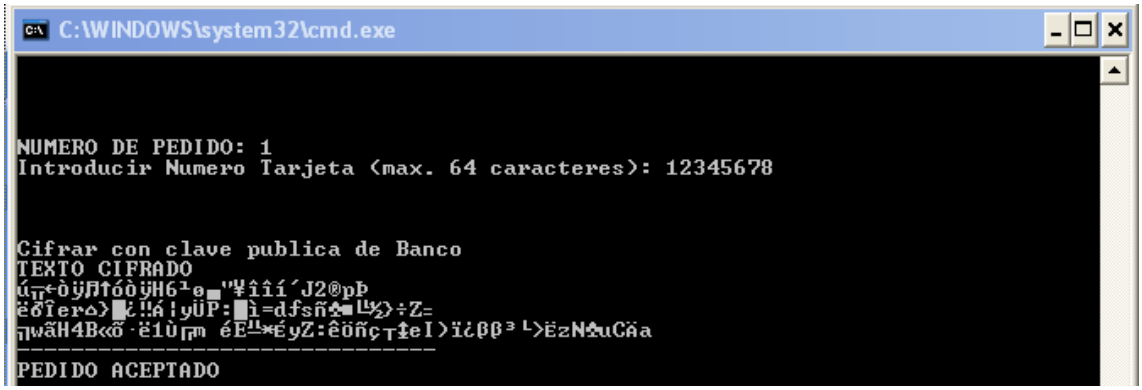
# 5. Resultados

De cara a poder con mayo claridad los aspectos comentados a lo largo de la memoria, en este punto se mostrará la traza de la ejecución del proyecto.

## 5.1 Cliente

En esta primera imagen, vemos como el usuario ha metido los datos de la tienda y el producto, se crea el socket SSL y muestra el certificado de la Tienda(Propietario del Certificado) firmado por la CA creada(Emisor del Certificado).

A continuación en la traza del Cliente podemos observar como recibe el número de pedido, posterior a ello introduce el número de tarjeta, que será cifrado para mantener la confidencialidad de estos datos. Si toda la transacción ha ido correctamente, llegará el mensaje de “Pedido Aceptado”.



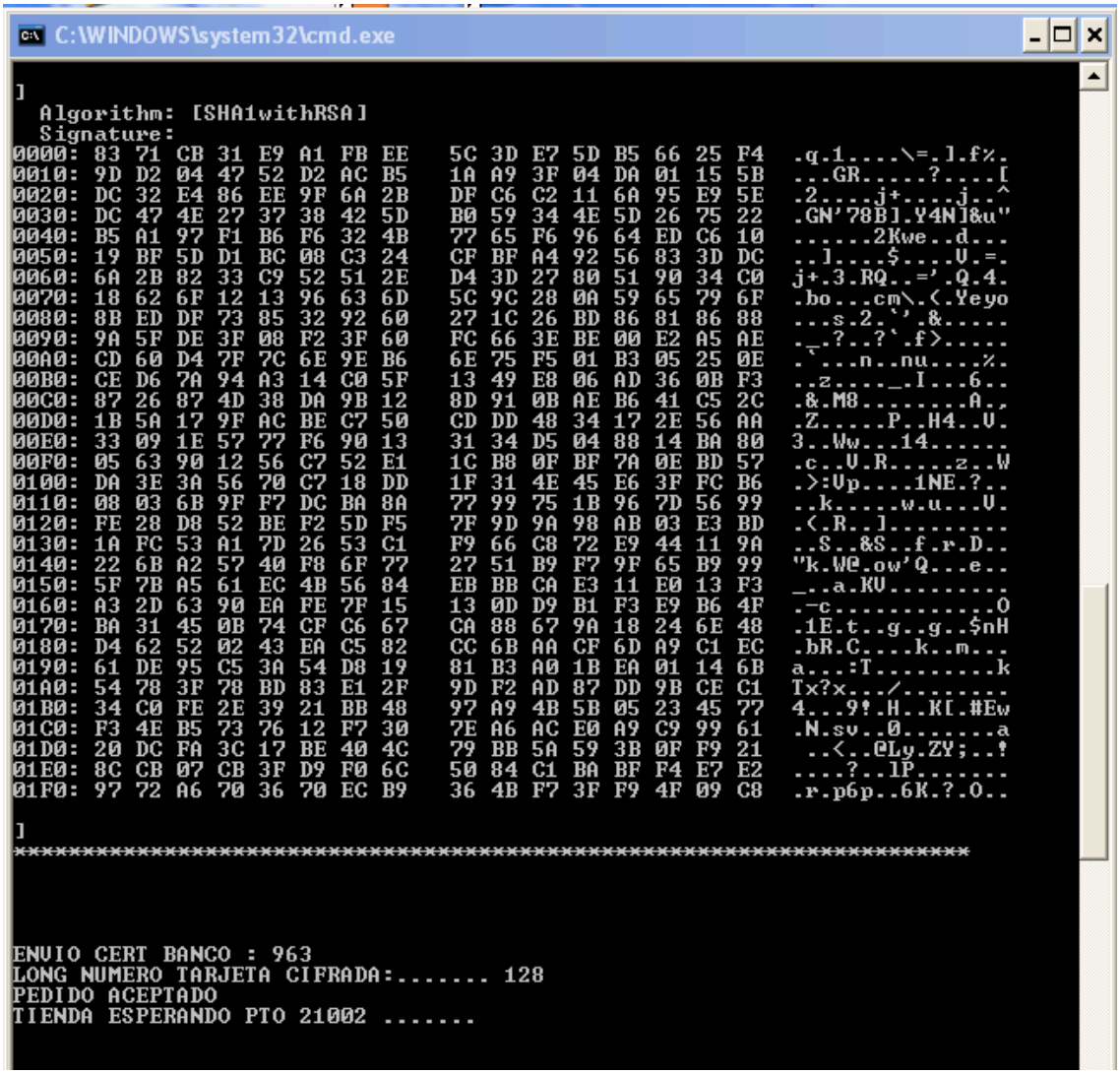
## 5.2 Tienda

Ahora pasamos a ver la traza de la entidad Tienda, en primer lugar vemos como esta esperando a la conexión de algún cliente en el puerto que hemos utilizado en el proyecto que es el 21002.

Tras la conexión de algún cliente (Propietario), vemos su certificado, al estar firmado por una CA (Emisor) en la que confía , autentica y acepta la conexión.

El siguiente pantallazo, muestra como le llega a la tienda el producto solicitado por el Cliente, en este caso 123, y tras ello ponemos el puerto del Banco y conectamos con él, recibiendo su certificado.

Por último, enviamos al banco la tarjeta de crédito del cliente, para asegurar su confidencialidad, a la tienda le llega cifrado, y esta entidad solo tiene como función enviar dicho texto cifrado al banco.

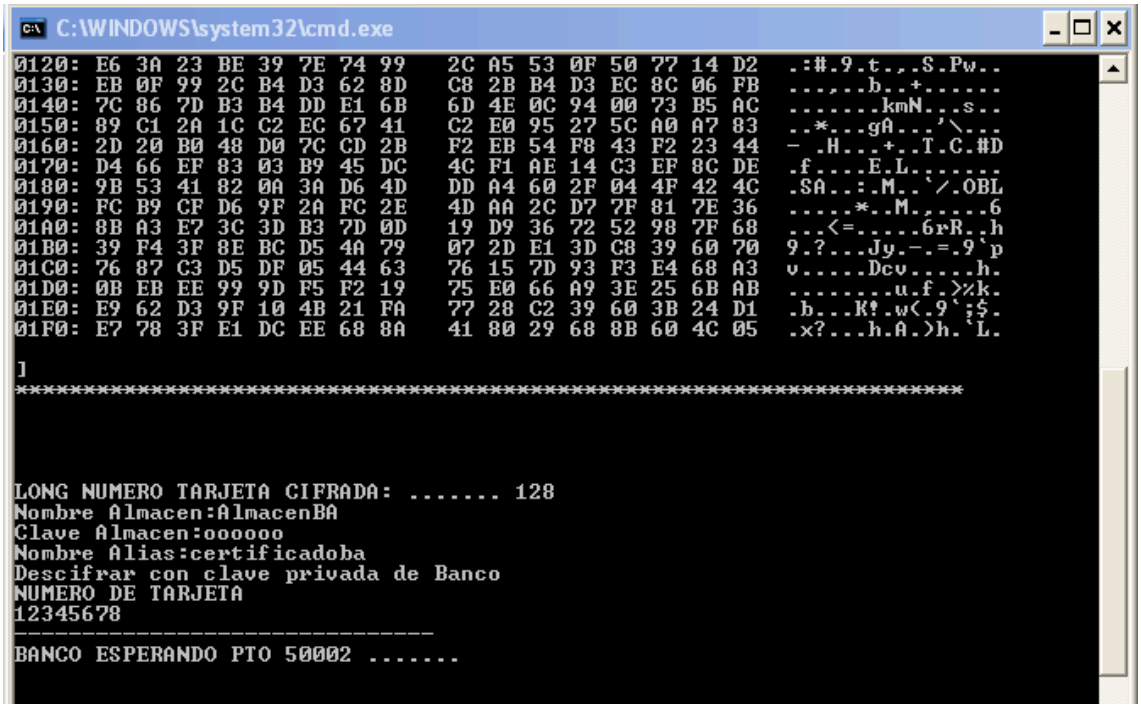


## 5.3 Banco

En la traza del Banco vemos como esta esperando a la conexión de alguna tienda en el puerto que hemos utilizado en el proyecto que es el 50002.

Tras la conexión de alguna tienda(Propietario) , vemos su certificado, al estar firmado por una CA(Emisor) en la que confía, autentica y acepta la conexión.

El siguiente pantallazo, muestra como le llega al Banco el número de tarjeta del Cliente, lo descifra con la clave privada y vemos que concuerda con el insertado en el Cliente. Ya podría realizar la transacción económica.



## 6. Bibliografía

Java Network Programming, 4º Edition. E. Rusty Harol, 0`really. 2013. http://www.it-ebooks.info/book/3137/.

Cryptography and Network Security Principles and Practice Fifth Edition. W. Stallings 2011, Pearson Education, Inc., publishing as Prentice Hall http://faculty.mu.edu.sa/public/uploads/1360993259.0858Cryptography%20and%20Network%20Security%20Principles%20and%20Practice,%205th%20Edition.pdf

Network Security with OpenSSL. J. Viega, M. Messier, P. Chandra. 0`really 2002 http://it-ebooks.info/book/263/

## ANEXO 1. Librerías JAVA

Podemos decir que la seguridad en java se concentra en tres extensiones básicas:

**JCE, Java Cryptography Extension**: Proporciona facilidades de cifrado, resúmenes (funciones hash), mecanismo de firma y gestión de claves.

**JSSE, Java Secure Sockets Extension**. El API de JSSE define un conjunto de clases utilizadas para ejecutar las operaciones necesarias para invocar el protocolo SSL desde el código fuente de java. Estas clases serán una extensión de las conocidas clases de sockets tradicionales.

**JAAS, Java Authentivation and Authorization Service :** Proporciona autenticación dentro de la plataforma Java. Todas las facilidades principales del diseño de la seguridad en Java están pensadas para proteger a los usuarios finales de las influencias de los desarrolladores: los usuarios finales dan permiso a los desarrolladores a acceder a los recursos en la máquina del usuarios final. JAAS permite a los desarrolladores obtener (o no) el acceso a sus programas basándose en las credenciales de autenticación proporcionadas por el usuario.
Nuestro interés se centrará en las dos primeras extensiones que nos permitirán construir aplicaciones de usuario en un entorno distribuido incorporando mecanismos y servicios de seguridad.

## ANEXO 2. OpenSSL

OpenSSL es un entorno integrado que permite la creación y gestión de certificados digitales. OpenSSL dispone de la infraestructura necesaria para crear una Autoridad de Certificación, firmar certificados de usuario, revocar certificados etc .

OpenSSL es además un paquete de herramientas de administración y librerías de seguridad para la gestión de certificados y la implementación de mecanismos de seguridad. OpenSSL proporciona un entorno mediante el cual es posible mediante línea de comandos crear certificados, firmas digitales, cifrado/descrifrado información, mensajes S/mime etc….Además proporciona librerías para implementar aplicaciones de seguridad en entorno C/C++ en sistemas operativos windows/Linux.

Esta librería criptográfica surge a partir del proyecto SSLeay, que fue iniciado por Eric A.Young y Tim J.Houston, y como propuesta inicial pretendía ofrecer el protocolo SSL para cualquier aplicación de seguridad. El proyecto OpenSSL fue iniciado en 1995, y actualmente es la librería criptográfica más usada. Algunas de las razones que la hacen tan popular son:

- Es distribuida, usando una licencia de código libre. Esto permite a los usuarios realizar cambios y optimizaciones constantes en los elementos criptográficos que la componen.

- Es posible integrarla en cualquier aplicación comercial sin necesidad de abonar tasas, permitiendo ofrecer soporte SSL criptográfico, comparable con casi cualquier librería comercial. Puede ser ejecutada en numerosas plataformas, permitiendo la exportación de las aplicaciones que la utilizan a cualquier sistema operativo.

## ANEXO 3. Keytool

Keytool es una herramienta de gestión de certificados y claves. Permite a los usuarios crear y administrar su propio par de claves pública/privada y el certificado asociado para autenticarse frente a otros usuarios e implementar mecanismos y servicios de seguridad.

Keytool también gestiona los certificados de las otras entidades, así como solicitudes de firma de certificados por Autoridades de Certificación reconocidas.

Además la herramienta almacena las claves y los certificados en un llamado almacén (keystore). La implementación por defecto del almacén es un fichero que protege las claves privadas con una password.

## ANEXO 4. Protocolo SSL

El protocolo SSL (Secure Sockets Layer) es el protocolo de seguridad más extendido en Internet. La inmensa mayoría de aplicaciones en la red utilizan este protocolo para proporcionar servicios de seguridad.

El protocolo SSL siempre requiere que el servidor envíe su certificado. Este certificado permitirá crear un canal seguro. El cliente cifra la información con una clave de sesión que es enviada al servidor cifrada con la clave pública del servidor extraída de su certificado.

El servidor opcionalmente puede exigir al cliente que se autentifique. En este caso el cliente está obligado a enviar un certificado digital y una firma electrónica.

El protocolo de seguridad SSL (Secure Sockets Layer) es bien conocido. Consta de 4 fases de intercambio:

En la primera fase el cliente y el servidor negocian la familia de cifradores a utilizar en el intercambio.

En la segunda fase el servidor envía un certificado de clave pública con el que autenticarse frente al cliente. El cliente se asegurará que ese certificado recibido es fiable y además protege el sitio web al que trata de conectarse el cliente. Opcionalmente el servidor pude solicitar en esta fase al cliente que envíe un certificado para que se autentifique.

En la tercera fase el cliente utiliza el certificado de clave pública del servidor para cifrar las claves de sesión con la clave pública del servidor garantizando de este modo la confidencialidad en la distribución de las claves.
Si el servidor solicitó al cliente autenticación en la fase anterior, el cliente está obligado en esta fase a enviar un certificado de clave pública firmado por una entidad que para el servidor sea fiable…..Además el cliente hará una firma de los mensajes.

En la ultima fase se actualizan los cifradores negociados y se envía el primer mensaje cifrado y autenticado con las claves acordadas

## ANEXO 5. Guía de Usuario

De cara a facilitar el uso de este proyecto, a continuación se detalla los pasos para poner en funcionamiento la práctica:

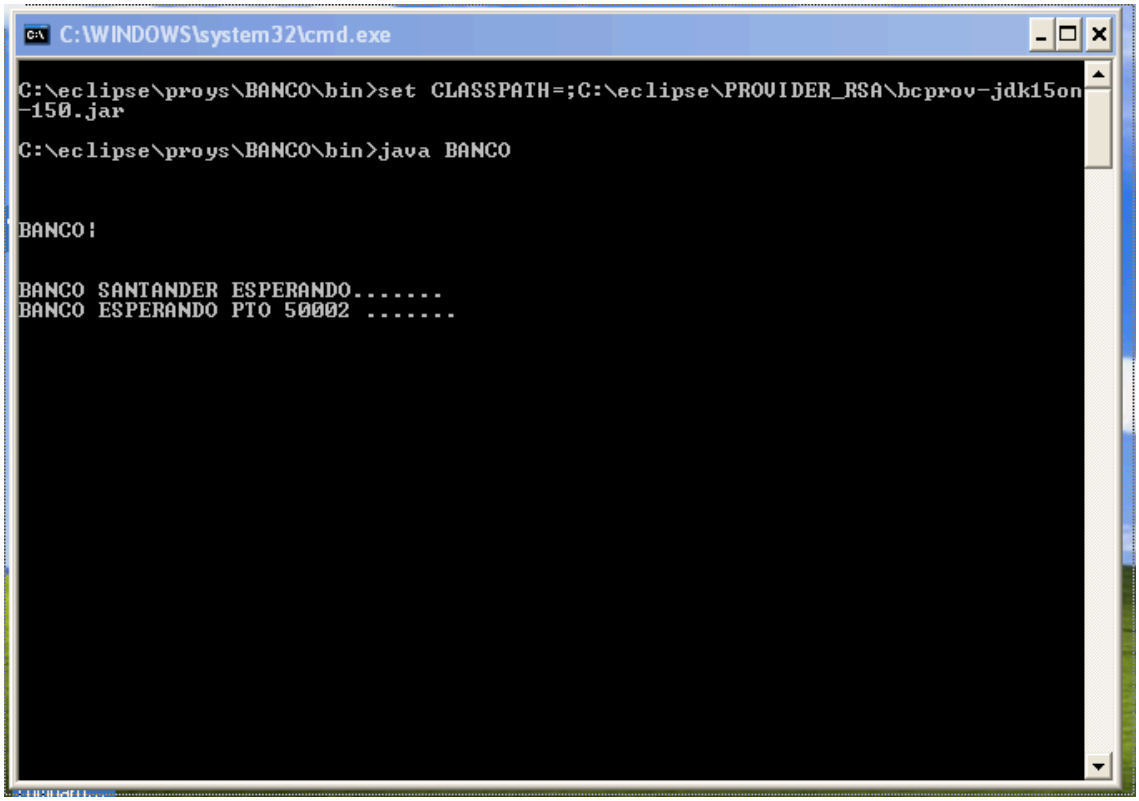
1. En primer lugar hay que ubicarse en la carpeta bin de los tres proyectos, Banco, Tienda y Cliente.

    - Por ejemplo , C:\eclipse\proys\BANCO\bin .

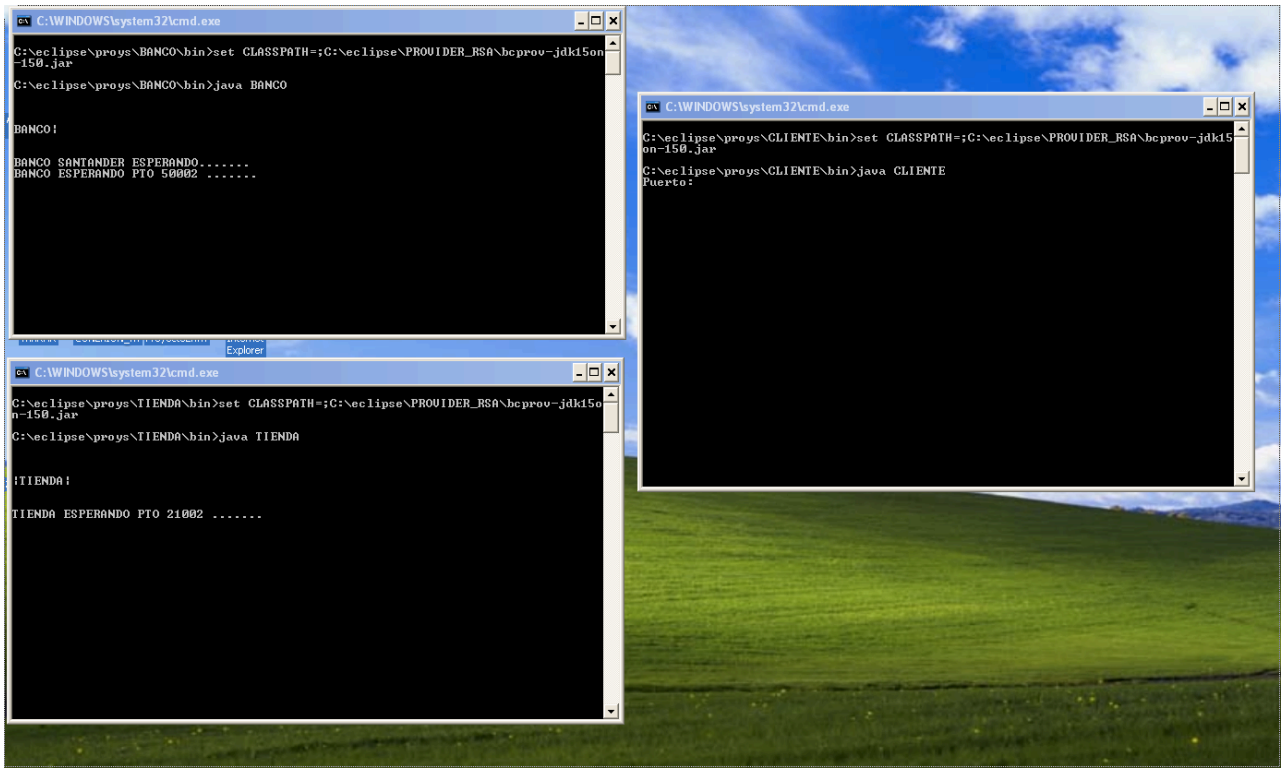
2. Una vez hecho esto, en cada carpeta bin tendremos:

    - El .Class de cada proyecto
    - Los Almacenes necesarios para la gestión de los certificados, las autoridades de certificación y sus claves.
    - Un fichero ejecutable , en Windows un .bat, que ejecutará cada proyecto.
        - Su nombre será BAT_ARRANQUE_NOMBREENTIDAD.bat

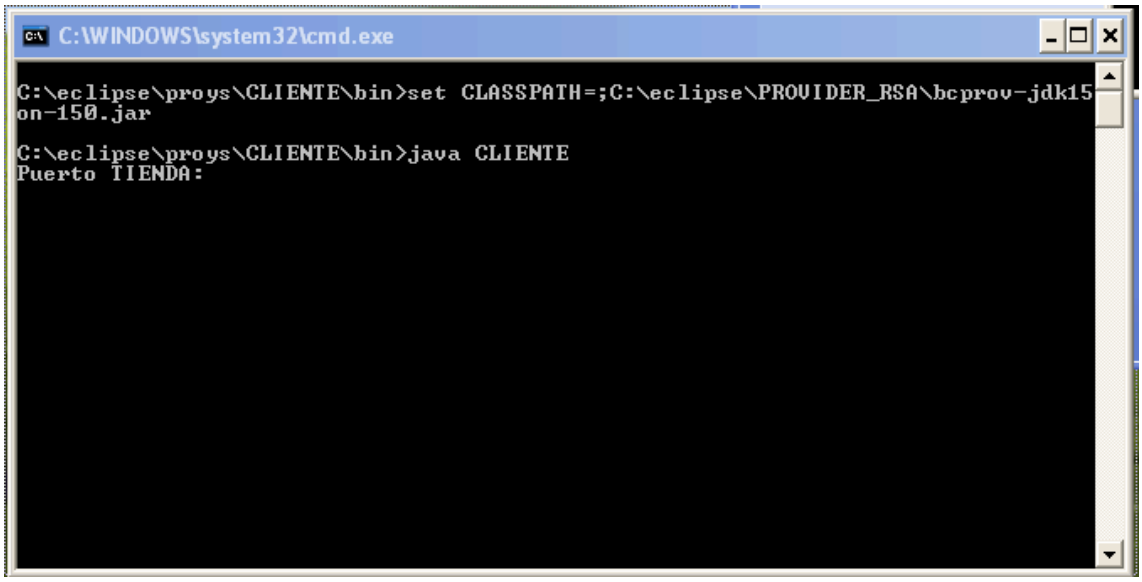
3. Hay que ejecutar el .bat, para el caso del Banco saldría la siguiente pantalla.



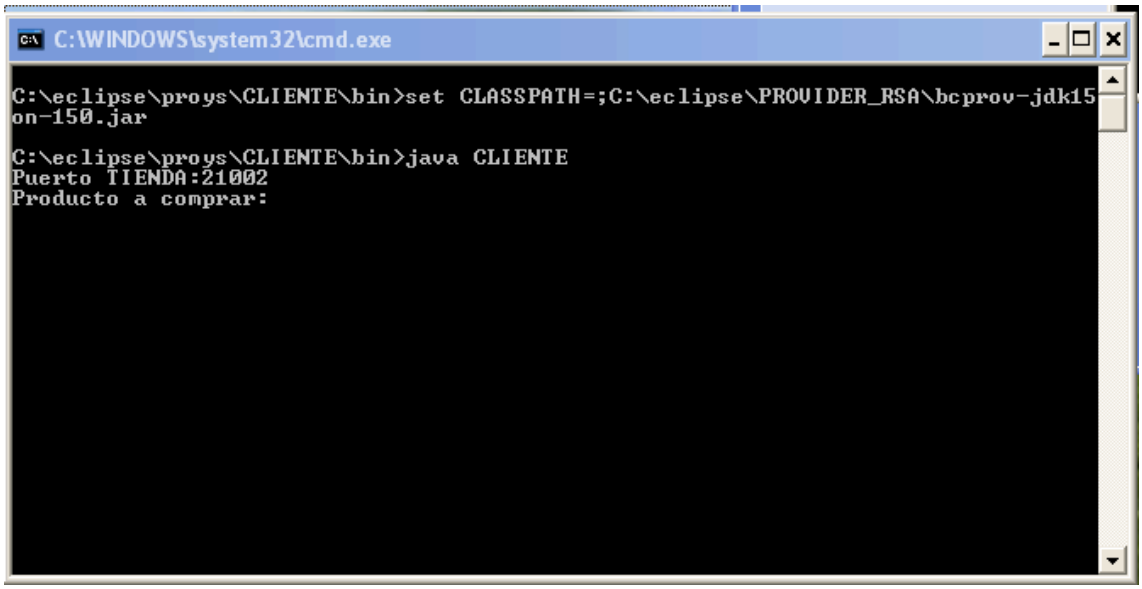
4. El siguiente paso habría que ejecutar los otros dos ficheros ejecutables. Quedando las tres líneas de comando abiertas.



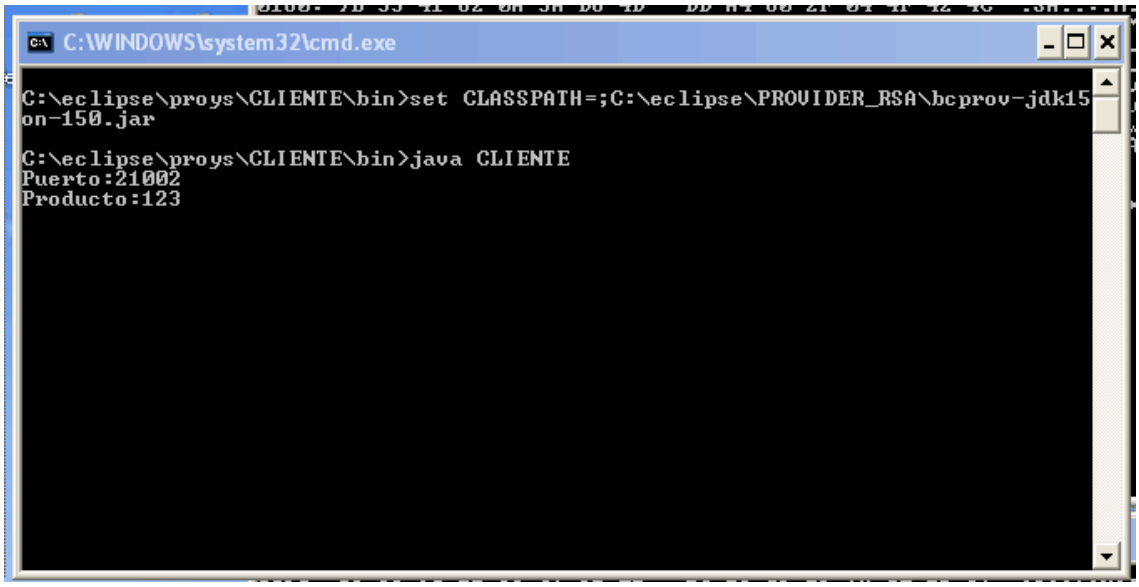
5. Una vez estén arrancadas las tres entidades, vamos a la entidad Cliente. Allí encontramos que nos solicita el Puerto de la tienda:



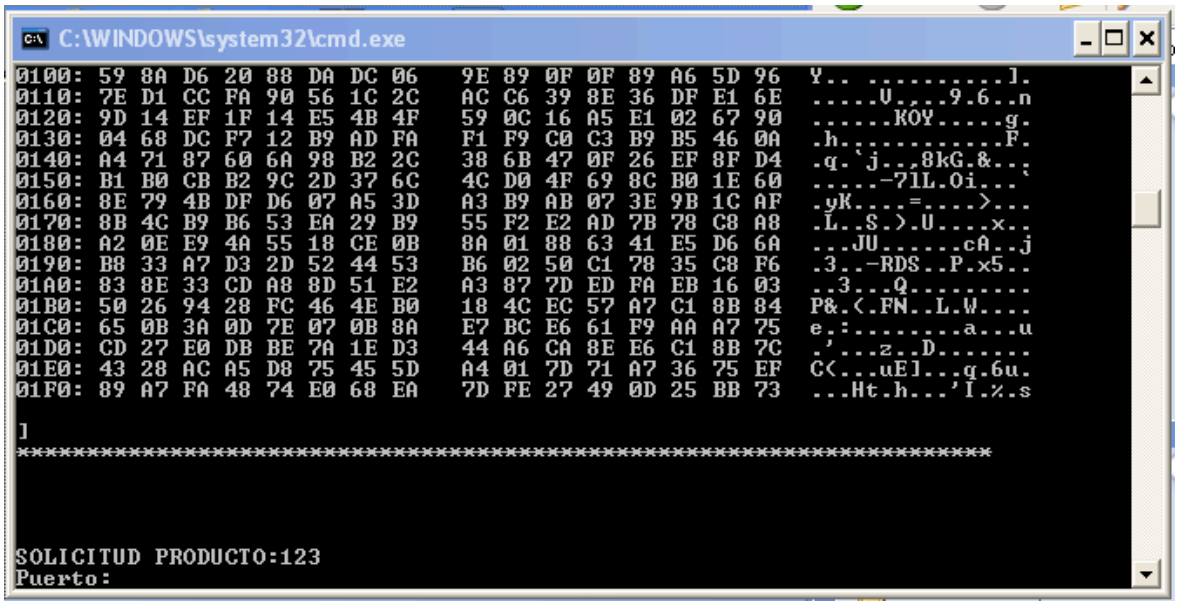

6. El puerto para conectar con la tienda es el 21002, vemos que una vez puesto el puerto y pulsando en la tecla ENTER, podremos indicar el producto a comprar.



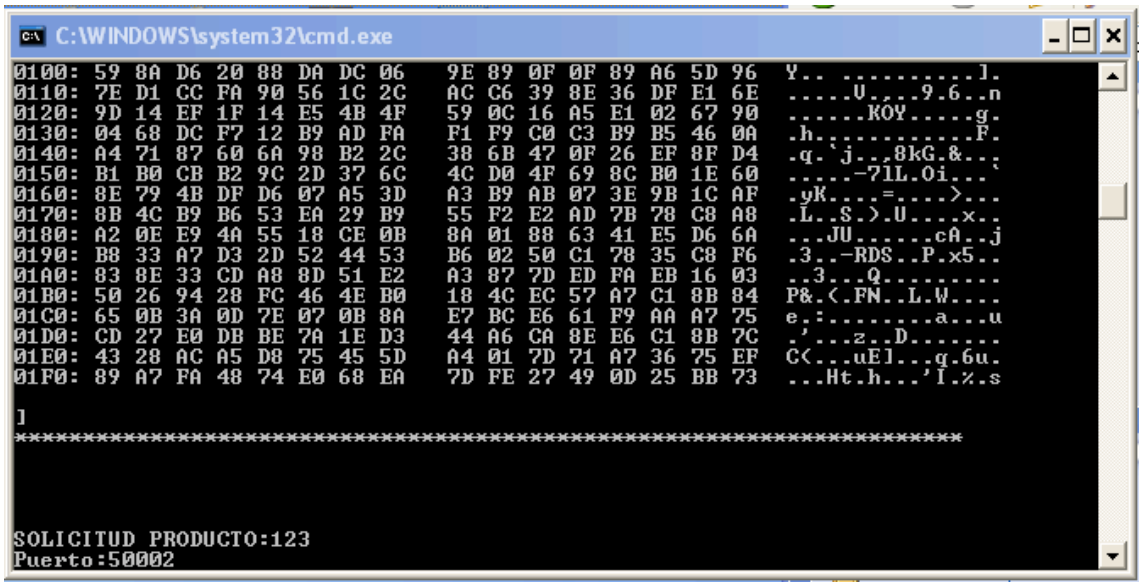
7. Al igual que hicimos con el Puerto, lo haremos con el Producto, insertamos el que queramos comprar, para este ejemplo será el 123, y pulsamos ENTER. Ya con esto conectamos con la TIENDA



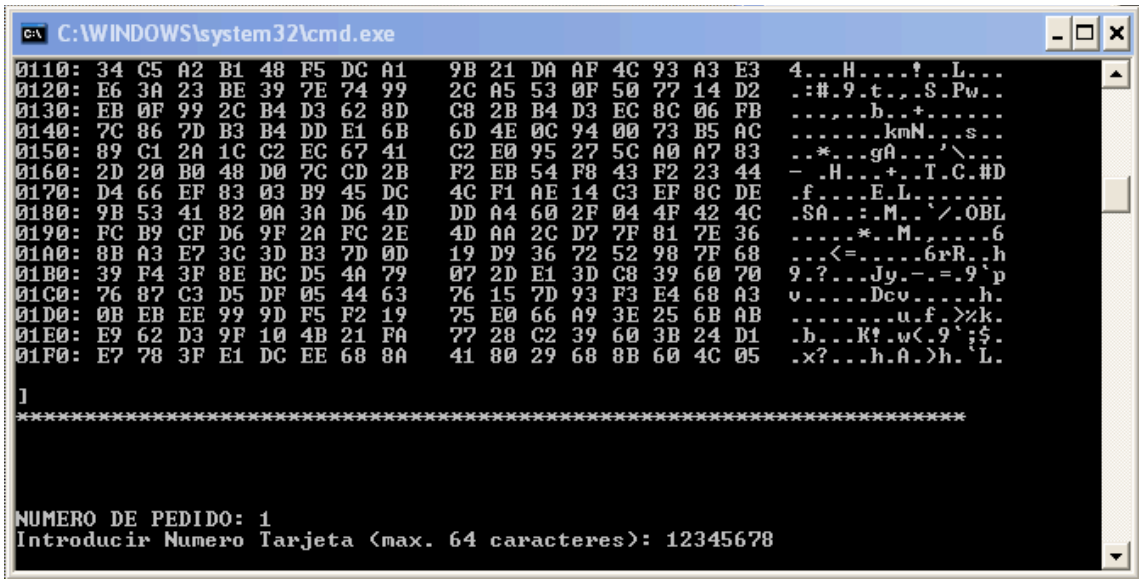

8. Ahora nos situamos sobre la entidad Tienda, donde nos llegará el producto comprado, anteriormente seleccionado por el Cliente. El siguiente paso será conectar con el Banco, para ello tendremos que escribir el puerto correspondiente.



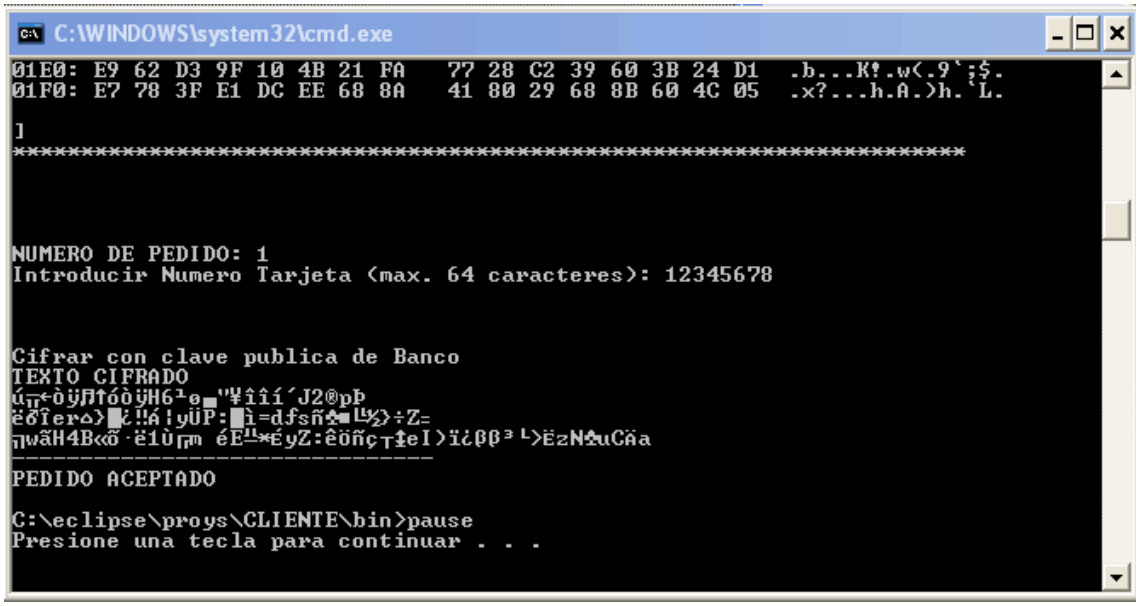
9. Escribimos el puerto para conectar al BANCO, en este caso será el 50002, y pulsamos ENTER, conectándonos .



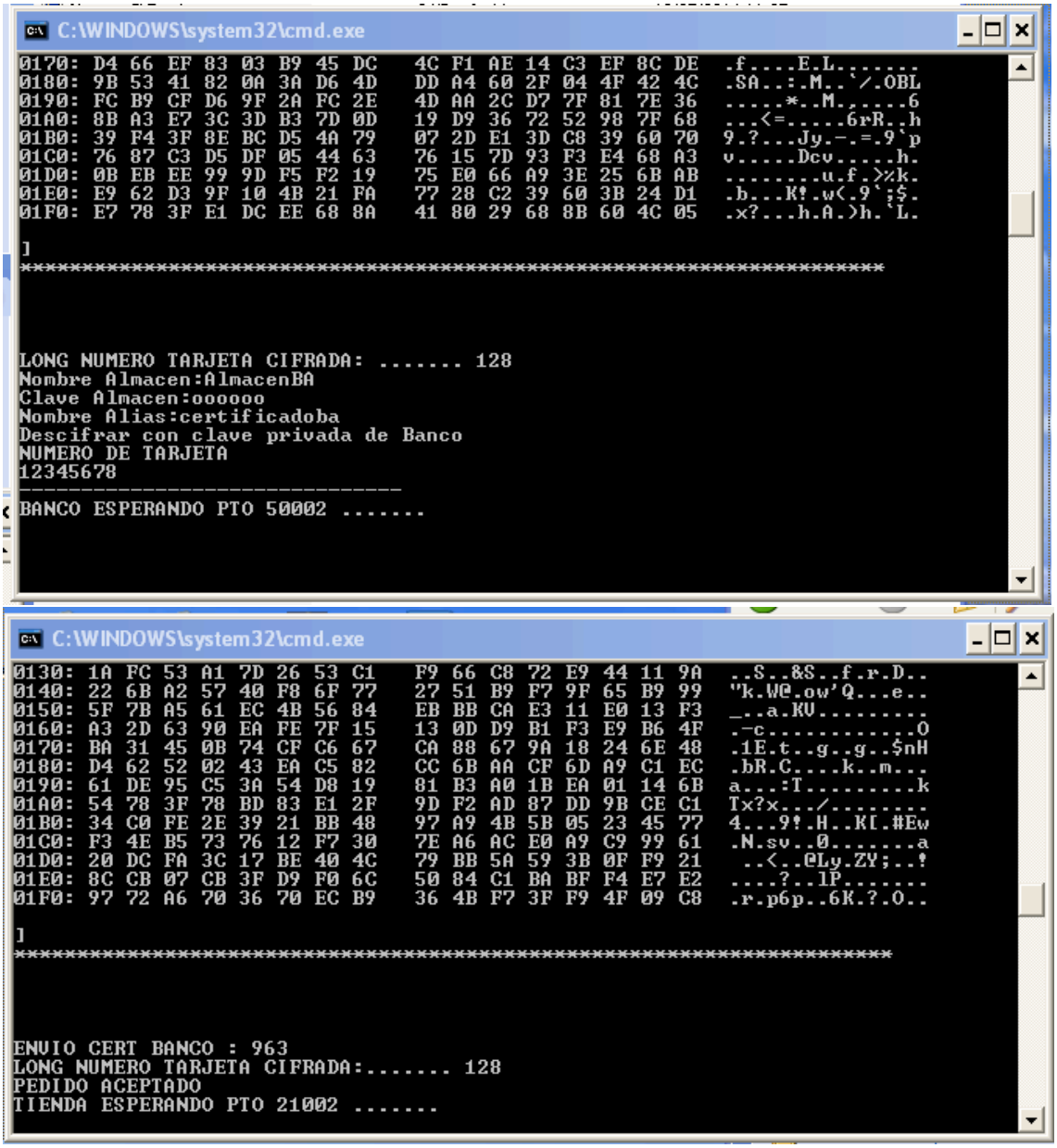

10. El siguiente paso , será que el cliente escriba el número de la tarjeta, para ello volvemos a la entidad Cliente, donde vemos que ya tenemos el Número de Pedido y ya podremos escribir dicha tarjeta. Para este ejemplo utilizaremos la 12345678.



11. Finalmente, vemos que en el Cliente una vez insertado el numero de tarjeta, lo cifra y muestra el texto cifrado, y en el caso de que todo haya ido bien que el pedido ha sido aceptado.



12. Por último, vemos que ha ido bien la transacción en la Tienda(Pantalla Inferior) y que se reinicia para esperar a nuevos clientes.
    A su vez el banco , vemos que le llega el número de tarjeta correctamente tras descifrarlo y se reinicia para esperar otras tiendas.



Este documento esta firmado por

| | | |
|---|---|---|
|  | **Firmante** | CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES |
| | **Fecha/Hora** | Fri Jun 06 18:35:18 CEST 2014 |
| | **Emisor del Certificado** | EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES |
| | **Numero de Serie** | 630 |
| | **Metodo** | urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature) |